ORION Instruments

Z80 Tutorial for the UniLab II™

Z80

ORION Instruments, Incorporated
702 Marshall Street
Redwood City, California
94063

Contents

INTRODUCTION

The UniLab opens an interactive window into your development system. Developers of microprocessor-based devices, who are already accustomed to traditional debugging, can just as easily monitor their systems' real-time performance. You can execute and fully test your programs before embedding them in ROM, even before the target hardware is complete.

The powerful UniLab concept offers transparent emulation, non-intrusive analysis, and precise specification of execution cycles. All target activity can be monitored, including bus signals, port I/O, register contents, target RAM, and emulation ROM. Prototyping, debugging, and optimizing are all done from your IBM-compatible computer.

Take the time to explore all of the UniLab functions. The analyzer, trigger, debug, assembly, and memory facilities work together seamlessly to greatly increase your productivity.
[figure 1-1]

What The Tutorial Covers

This tutorial introduces the major UniLab components, providing a basic understanding of the software and specific Z80 operations. Included is a simple program that runs on the Orion Micro Target. In the course of exploring that program's operation, you will exercise the UniLab thoroughly. Follow the step-by-step instructions, and it will be easy to learn all the related commands and functions. You will use the system productively in a very short time.

How to Use The Tutorial

Be sure to sit within easy reach of your computer when you use the tutorial. Follow the instructions for connecting it to the UniLab. Then, every time you open the tutorial:

- First, turn on the system.
- Read a section, doing all the exercises.
- Experiment with the material you learned.
- Take a break before proceeding further.

We suggest a rest after each chapter, to let the material sink in and to avoid information overload. It is best to proceed sequentially, from beginning to ending, but feel free to explore the system as your interests dictate.

What You Need to Proceed

All of the UniLab's resources can be tapped immediately, without complex or costly add-ons. Here is all you need:

- IBM PC (or compatible) computer with:
  - two floppy disk drives (or optional hard disk)
  - 320K RAM
  - one standard serial port
  - DOS version 2.0 or higher
- The UniLab unit
- Z80 Personality Pak (includes microtarget with emulation module, and software).

Get Connected

Setting up the UniLab system is easy. The hardware is ready to plug into any IBM PC-compatible host computer. The software must be backed up onto your system disk. Then you are ready to become familiar with the powerful new tools at your disposal.

Install the Hardware

First, unpack the UniLab. Two cables emerge from the back. Attach the one with an RS-232 connector to the host computer's serial port (it uses a standard, 25-pin serial port; if the host port has only 9 pins, you will need to get an adapter). Plug the other cable into a power outlet.
[figure 1-4a]
Now open the Personality Pak and look at the microtarget board, leaving it in the protective box. The smaller, elevated board holding the Z80 CPU is called the emulation module. Check to see that it is still firmly seated on the microtarget after shipping. The clearly labelled power jumper wire provides power to the board, and would be disconnected if the emulation module were used on another board with its own power supply.
[figure 1-4b]
Handle the board as carefully as any other electronic circuitry: keep it clean, keep liquids safely away, and avoid impacts or pressure on the various components. Do not allow anything to cause an electrical short between the pins or other conductors.

Two ribbon cables come off the emulation module. Their free ends are labeled "emulator cable" and "analyzer cable."

Plug the **emulator** cable into the UniLab front panel connector marked "8/16 Bit In-Circuit Emulator." A small, plastic "key" indicates the top side of the cable. Taking care not to bend any of the pins, press evenly until the cable is seated snugly in the connector. In the same way, plug the **analyzer** cable into the "48 Channel Bus State Analyzer."
[figure 1-4c] [figure 1-4d]
Now you have finished installing the UniLab hardware. Turn on the power switch on the front panel of the UniLab: the indicator light confirms that the unit is turned on.

Install the Software

(For installation on a hard disk, please refer to the "Software Installation" insert that accompanies the UniLab diskettes.)

1. Make a system diskette.
   Boot the host computer with a copy of the DOS version (3.2 or higher) you will use with the working UniLab system. Insert a blank diskette into drive B. Format it as a "bootable" system diskette by using the **FORMAT B: /S** command.

2. Copy all UniLab files to the new diskette.
   Put away the original DOS diskette and insert the master UniLab Main Program diskette into drive A. Copy all the UniLab files onto the new diskette in drive B by using the **COPY A:*.* B:** command.

3. This is also a good time to make a backup copy of the UniLab Glossary diskette (do <u>not</u> put the glossary files on a system diskette).

4. Now put the master UniLab diskettes safely away.

5. Put the new, bootable copy of the UniLab Main Program disk in drive A. The copy of the UniLab Glossary should be in drive B, where it will be used for on-line help functions.

If you encounter any difficulty with these procedures, please refer to your DOS manual.

THE UNILAB ENVIRONMENT

Master the User Interface

Start the UniLab system
Get help with commands
Use menus & function keys
Leave the program

EXPLORE THE UNILAB ENVIRONMENT

The UniLab environment provides immediate access to the entire development system. Analyzer, debug, assembly, and memory functions are co-resident at all times. An easy way to become familiar with the general capabilities of the UniLab is to browse through its menus. You will see the various components of the software system and their relationship to both the target and UniLab hardware. Later, detailed investigation will fill in the overall picture you gain in this section.

Start the UniLab System

Turn on the UniLab power switch. Put your working copy of the UniLab system diskette in drive A and the glossary diskette in drive B, then boot the host computer.

At the A> DOS prompt, type **ULZ80** to execute the UniLab software.

Checking for Problems?

If the last word on the screen is "Initializing..." and your keyboard seems frozen, be sure the red indicator light on the front of the UniLab indicates that the power switch is turned on. Also check that all cables are snugly attached. Then press **Ctrl-Break** to free the keyboard, and type **INIT**. Communication with the UniLab may be disrupted by devices connected to other RS-232 ports. If trouble occurs, try connecting the UniLab to a different RS-232 port. If it persists, see the UniLab Reference Manual for "Troubleshooting."

Enter Commands

The program starts out in command mode. The introductory display provides general information about on-line help and the menu mode of operation. A cursor indicates where commands may be entered.
[figure 2-4b]
Try typing **WORDS** followed by a carriage return.

An alphabetical list of all UniLab commands is displayed with brief usage notes. The concluding "PgDn for more" always appears when more data can be displayed by pressing that key. Try it, or press the Enter (carriage return) key to end the listing. When the listing terminates, the "ok" prompt and a cursor are displayed.

Enter a misspelling: type **WIRDS** and a carriage return.
A "not recognized" message appears and the incorrect entry is underscored.

Get Help On-Line

The UniLab software will provide detailed assistance when you need it.

Type **HELP BINLOAD** followed by a carriage return.
A screenful of information is presented about the BINLOAD command: what it does, its usage, and an example.

Now try a misspelling: **HELP WIRDS**, for instance.
The "WIRDS not in glossary" reply indicates that help is not available for that subject. On-line help is available only for commands UniLab recognizes.

Enter the single word **HELP**.
The resulting text explains how to get assistance with specific commands and with command-mode operations assigned to the function keys. This is the same introductory screen that appears every time you start the program with **ULZ80**.
[figure 1-7]

Select Functions by Menu

This tutorial relies heavily on the UniLab menus. With them, there is no need to learn a lot of commands before using the system – function keys offer easy access to most operations. Later, you will find the command mode more expedient for many operations.

Press **F10** to enter the menu mode of operation.

The "UniLab Main Menu" is displayed on your screen. The menu presents a list of choices and the function key used to select each one.

[figure 2-4a]

Selecting any menu item, except F10, will display a sub-menu.

Press **F8** to examine the "Toolkit Routines."

The sub-menu presents a list of choices.

Press **F10** to return to the main menu.

Press F10 at any sub-menu to return to the main menu.

Leave the Program

When you are ready to stop work, type **BYE**. You will be left at the normal DOS prompt. When working with the UniLab software, a system reset or power-down is safe at any time.

USE THE BUS-STATE ANALYZER

Observe target execution without interfering

Analyze Program Execution

Enable memory and load a program

Disassemble emulation ROM

Capture and display a trace

Scroll the trace display

Interpret the display

Choose Other Analyzer Options

Sample cycles while executing

Use the Bus-State Analzyer

Developing microprocessor-based devices goes hand-in-hand with debugging, the most time-consuming phase of many projects. Interrupting the target processor and single-stepping through its instructions until a problem is detected can be laborious and, for certain problems, very inefficient. UniLab captures information about the real-time execution of your target system, allowing you to track down bugs and test hypotheses quickly.

In this chapter, you will observe a program running on the microtarget. Bus cycles will be recorded in the trace buffer and displayed to the user: all program instructions and their effects will be monitored without disturbing target execution.

Load a Sample Program

Boot the UniLab system and enter menu mode by pressing **F10**.

Press **F2** to use the "Load or Save a Program" sub-menu.
[figure 3-3a
Select **F2** to load a binary object file from disk.

UniLab requests the memory address where you wish to start loading the file. Enter **0** to start loading at address 0000. UniLab now requests an ending address for the program.

Since you don't know the length of the sample file, just enter **FFFF.**

UniLab will stop loading a program at the ending address you specify or at the end-of-file, whichever it reaches first. After you enter the ending address, the program displays the commands you would type if you weren't using the menus, and requests the name of the file you wish to load.

Type **DEMO3.TSK** for the filename.

UniLab loads DEMO3.TSK and shows the program's ending address.

Press **F10** to return to the main menu.

Disassemble Emulation ROM

Since this is your first use of the system, before proceeding further you may wish to confirm that the program has been loaded.

Press **F3** for the "Examine and Change Program Memory" sub-menu.
	This menu provides several options we will explore later.
	[figure 3-3b]
For now, press **F2** to view a disassembly of the code.
	The program requests a starting address for the disassembly.

Enter **0** as the starting address.
	The program now asks for the number of lines you wish to disassemble — it will provide five lines by default, if no value is entered.

Press **Enter** to see five lines.
	The equivalent keyboard command is displayed, followed by the listing. The three columns represent addresses, bus data, and disassembled Z80 mnemonics. These will be discussed shortly — for now, we are just confirming that the target program is in memory.

Capture and Display a Trace

Return to the main menu by pressing **F10**.

Press **F4** to use the analyzer sub-menu.
[figure 3-2]
Press **F1** to display the initial cycles of program execution.

> This displays the equivalent keyboard command (STARTUP), then issues a reset to the target system, and records initial bus activity.

[figure 3-4]
The display shows what took place during the first cycles of the DEMO3.TSK program's execution on the target system. All bus activity was recorded until the analyzer's trace buffer was filled, then the buffer's contents were uploaded to the host computer. The microtarget's LEDs reveal that the target system continues to execute the program even after the analyzer display is complete.

Scroll the Trace Display

More data is obtained than can be shown on one CRT screen – the UniLab trace buffer holds 170 (AA hex) target cycles.

Use the **Down-Arrow** and **PgDn** keys to look through the trace.

> (The up-arrow and PgUp keys scroll back through the CRT history, not through the trace buffer's contents.)

Press the **Home** key to return to the beginning of the trace.

Interpret the Display

The columns of data displayed by the analyzer are shown below:

The first line of our sample trace tells us this: The 0 in the cy# column shows this is the first cycle of the trace — called the "trigger" cycle. We will disregard the CONT column in this tutorial (see chapter 6 of the UniLab Reference Manual for details). The ADR column shows that execution during this bus cycle was at address 0000. DATA column contents 31FE18 are disassembled to the right: load into the stack pointer the value 18FE. The HDATA and MISC columns display input from the UniLab's additional signal wires; their values float high when unused (these columns can be eliminated from the display with the mode panel, discussed later).
[figure 3-5]
Refer to the next line in the trace display. From the cycle number, we see that it took three bus cycles to fetch the previous instruction.

Note: if DATA contains a two-byte immediate value, the bytes are shown in reverse order (Intel format).

Choose Other Analyzer Options

So far, you have traced the first AA (hex) bus cycles of the target program's execution. Some menu options illustrate other simple uses of the analyzer. You can try them now, while you are in the analyzer sub-menu.

F2 — displays the most recent 170 cycles when you press the key.
F5 — samples about two bus cycles per second. Press any key to stop.
F6 — samples about two address references per second. Press any key to stop.

The UniLab can also show what happens to the target system after (or before) a particular event takes place. And when analyzing execution, it is often useful to record only certain types of bus cycles. The next chapter shows how to observe any part of a target program's activity. You will use more sophisticated UniLab functions to understand DEMO3.TSK's execution on the microtarget.

Build Analyzer Triggers

UniLab allows you to define triggers that recognize target-board bus events of any type, and to start (or end) a trace at that point. This makes it easy to observe a program's behavior after a particular data value, range of addresses, or other user-defined event appears on the bus. You can record all bus cycles, or only cycles that match a trigger description.

The powerful capabilities of triggers deserve close inspection, beginning with simple trigger specifications and progressing to more complex examples that use filters and qualifiers. Here we will use them to understand more of DEMO3.TSK's execution on the microtarget.

Inspect Current Trigger Status

With the sample program loaded, press **F1** at the analyzer sub-menu (the STARTUP command), which sets a default trigger and captures a trace.

Type **TSTAT** to view the current trigger status.

> This command displays the current trigger specifications. RESET indicates the target system will be reset when the analyzer is started. The DCYCLES value is the hex number of bus cycles the analyzer will record after it detects an event that matches the triggers specs. 0 QUALIFIERS indicates that no qualifiers (discussed later) are being used.
> [figure 4-2a]

Use Preset Triggers

When defining analyzer triggers, you will frequently use one of three "normalizing" words to clear out any previous trigger specifications. They differ only in the number of bus cycles they tell the analyzer to record after the trigger event (the DCYCLES variable).

Type **NORMB** followed by **TSTAT** to see the effect.

> The trigger status shows that the analyzer will record only 4 bus cycles after the trigger before "freezing"the trace buffer's contents.
> [figure 4-2b]

Try typing **NORMM** and **NORMT** and check their effects with **TSTAT**.

NORMB places the trigger event at the bottom of the trace buffer, preserving a

record mostly of pre-trigger cycles. NORMM places the trigger at the middle of the buffer. NORMT places it at the top of the buffer, recording the greatest number of post-trigger cycles.
[figure 4-3]

Define a Simple Trigger

The analyzer watches for an event that matches the current trigger specifications, and records the number of post-trigger cycles set by the NORMx words discussed above. To define an analyzer trigger, you just specify the characteristics of the desired target event.

Type **NORMT** to set the trigger near the top of the trace buffer.

Go to the analyzer menu and press **F1**.

> The display shows that during the first eleven cycles, the Z80 registers are initialized, followed by an LDIR instruction that transfers a block of memory from ROM to RAM.

Use **PgDn** to scroll the display through the repeated LDIR/read/write sequence.

> At cycle #94, the target system begins executing new instructions, starting at address 0016. But using PgDn again reveals that you have reached the end of the trace buffer. A trace that begins when address 0016 appears on the bus will better show what happens next.

Press **F3** to define an address as the analyzer's trigger. When UniLab prompts you to supply the address, enter **16**.

> The target system is reset (because RESET is enabled), and the analyzer begins looking for a bus cycle containing address 0016. When it appears, the trace buffer fills and its contents are displayed. The target system continues to execute.
> [figure 4-5a]
> You can see by the listing that the program loads a value into the A register and sends it to target port 7B. Next, another value is loaded into A, sent to port 79, and the value in A is rotated. Then the routine at address 00A0 is called. There, you can see that 40FF is loaded into the HL registers, then L is decremented in a conditional-jump timing loop.

Type **TSTAT.**

Your trigger spec (16 ADR) has been added to the trigger status – as long as it appears here; it is part of the current trigger specification. [figure 4-5b]

The three-byte instruction at 0016 jumps to 0082. Curiosity might lead us to find out what happens when the program reaches address 0019.

Use **F3** to trigger the analyzer when address 19 appears on the bus.

The analyzer displays a the Trigger Wait Status message while it waits.

Address 0019 never appears, so press any key to interrupt the search.

[figure 4-5c]

Use **TSTAT** again now.

The 19 ADR trigger spec replaced the previous 16 ADR. When you specify a new value for the same input group, it replaces the earlier one.

You can trigger a trace with a value in any of the displayed fields except the disassembled mnemonics and the cy# value (which is relative to the trigger event). Here is an example using data input:

Enter **NORMT 80 DATA S** to clear the old setting and start the trace when 80 appears in the data field.

The display shows a trace that begins (cycle #-1) with 80, the second byte in the data field, being loaded into the A register.

[figure 4-7]

Any byte in the data field can be used as a trigger. And while you cannot define a trigger with mnemonics, it is easy to use the opcode values in the data field if you want to trigger on a particular type of instruction. For example, 0F DATA will trigger on an RRCA instruction.

Extend Trigger Power

You can start a trace when any one of several values appear.

Enter **RESET'** so the system will trace the executing system without restarting it.

> Reset can be disabled with RESET' and enabled by typing RESET; or use the Mode Panel, discussed later.

Type **NORMB** to clear the previous trigger specifications.

Type **80 DATA ALSO 40 DATA S** to trigger the analyzer when either of those data values appears.

> This allows either data value to trigger the analyzer. As usual, UniLab displays the Trigger Wait Status line if it must wait for the trigger event. Press S repeatedly, and you will find that the value 40 frequently shows up on the data bus, in the timing loop at address 00A0. Less often, 80 triggers the trace when the contents of A are written to port 79.
>
> ALSO adds a value to the current trigger spec, and as many values can be added as you find useful; without it in the above example, 40 DATA would have overwritten the preceding 80 DATA.

There is an additional precaution when using ALSO with address input: The addresses are composed of two, eight-bit input values. Cross products will be produced if the upper bytes of ALSO-ed addresses do not match, and can cause a trigger on unanticipated values.

Trigger on a Range of Values

Now type **NORMT A0 TO A9 ADR S** to trigger the analyzer on a range of addresses.

> This address range contains the delay loop instructions. The trace begins with the first cycle that contains an address in the range A0 to A9. TO is most often used with address or data input.
>
> The purpose of these cycles is fairly self-evident, so it will be useful to trigger a trace when any other operation takes place.

Exclude Values from Trigger Specs

The previous example demonstrated how TO is used to specify a continuous range of values, and how ALSO includes multiple, non-continuous values in a trigger spec.

Many programs, including DEMO3.TSK, repeat one operation frequently during execution. You can instruct the analyzer to ignore that operation and trigger when any other event takes place. NOT is used to prevent single values, or ranges of values, from acting as triggers. This becomes valuable for trapping a bad address or bad data. In our example, it is used to trigger on target activity outside the delay loop.

First, type **NORMT S**.

Enter **NOT A0 TO A9 ADR S**.

> The trace begins with the first operation that does not contain an address in the range 00A0 through 00A9. Here it is reading the return address, which contains a jump to the program's main loop. This loop, at address 88, sends the value in A to the target port and rotates the contents of A before calling the delay loop at address 00A0 once again.

Remember that you can use TSTAT to see how UniLab interprets your trigger specifications.

Trigger on Multiple Fields

You have defined triggers using values from a single column of the analyzer display at a time. However, it is just as simple to incorporate several fields into your trigger specifications.

Enter **NORMB 79 LADR 80 DATA S**

DEMO3.TSK, as we saw earlier, sends the value 80 to the target port. Here, we specify a trigger with 79 in the low address byte and 80 as the data byte. The resulting trace shows the value 80 being output to port 79, followed by the rotate operation.
[figure 4-9]

Enter **NORMB 79 LADR NOT 80 DATA S** to see what else gets sent to port 79.

Cycle#0 contains a different value being output. Press S several times, and you will see the output values that result from rotating the contents of register A. These values determine the changing pattern displayed by the microtarget's LEDs.

Any combination of trigger specifications may be used:

- ALSO performs a logical "or," triggering the analyzer when one value or another appears in one of the input fields.
- TO extends that concept to any value in a range.
- NOT triggers on any value outside the specified number or range of numbers.
- Logical "and" operations occur when you define trigger specs for more than one of the input groups (e.g., when the desired trigger event contains a specific address and data value). All of the current specs must be met to trigger the analyzer.

Exercise a few, simple combinations of these commands before going on to learn further features. If no trace results after issuing the S command, it is because the trigger event you specified never occurs. A trigger's result may vary, depending on whether reset is enabled or disabled. Use the NORMx words to clear out your trigger specifications, as needed.

Filter the Trace

So far, you have learned how to trigger the UniLab analyzer's trace function. That is, you can begin a trace at any time by describing a specific target bus event. You can also specify the characteristics of the cycles you wish to examine, and only those will be saved in the trace buffer. With this conditional recording of target bus cycles, the display focuses only on the most valuable information.

The filter command clears any previous trigger specifications and will record A9 bus cycles that match the filter characteristics.

Type **ONLY 79 TO 90 LADR S** to record only cycles with a low address byte in the declared range.

> The Trigger Wait Status line appears while events matching the filter spec fill the trace buffer. Then the data is displayed; the "f" to the left of each row indicates that this was a filtered trace. These cycles represent the main program loop, including output to target port 79.
> [figure 4-10]

Enter **ONLY NOT A0 TO A9 ADR S.**

> This trace begins with the first cycle that falls outside the declared range of addresses, and all subsequent cycles in that range are filtered out of the trace. This trace is similar to the one obtained above, but includes read and write cycles generated by the RET and CALL instructions.
>
> (In the previous section, this same specification was preceded by NORMT instead of ONLY, and just defined the trigger event; post-trigger cycles were not filtered.)

Enable reset of the target system by typing **RESET**.

Type **ONLY WRITE S** to record only write operations.

> The first 20 lines in this trace are write cycles from the block-move operation at address 0014. The rest were generated by the CALL instruction. As the next example shows, you can exclude these initial operations from the trace.
>
> WRITE is a macro that specifies a preset range of CONT values. READ

and FETCH are macros that work similarly.

Qualify a Trigger Event

You can delay the analyzer's search for a trigger until after a qualifying event appears on the target bus.

Type **NORMT WRITE AFTER 181F ADR S.**

The analyzer waited until address 181F appeared on the bus, then started looking for the trigger (WRITE). Cycle #0 contains the first write cycle after the initial block-move operation. Use TSTAT to see how qualifiers, defined with AFTER, are noted in the trigger status.

Type **ONLY WRITE AFTER 181F ADR S.**

The filtered trace contains only write operations after address 181F appeared. These are all cycles that store a return address for the timing loop.

You could also trigger the analyzer at the beginning of the output loop after that loop has already executed once. In DEMO3.TSK, we know that after the 1 in A is first sent to target port (address 0088), A is rotated, and the delay loop at 00A0 is called before returning to 0088 to output the next value.

Type **NORMT 88 ADR AFTER A0 ADR S.**

The analyzer waits for the qualifier address 00A0 to appear on the bus, then triggers the trace with the next occurrence of address 0088.

[simple qualifiers chart from reference manual]

Up to three qualifiers can be used in a trigger specification. The qualifiers must occur in sequential cycles, starting with the last one entered, then the next-to-last one entered, etc. The trigger event can occur at any time after its qualifiers appear on the target bus.

USE THE DEBUG FEATURES

Establish Debug Control

Set a breakpoint
Read the breakpoint display
Single-step the target system
Resume execution to a new breakpoint
Follow jumps and loops

Alter Register Contents

Trigger a Breakpoint

Define debug triggers

Display the Alternate Registers

Read and Write to Target Ports

Use the Debug Features

The UniLab system provides full-featured debug facilities. You can set a breakpoint, or issue a non-maskable interrupt from the keyboard, to halt the target program and single-step through its execution. You can view and alter the contents of registers and memory, and use bus events to trigger breakpoints.

These tools are used while special debug hardware controls the target processor. A few bytes of emulation ROM (the reserved area and the overlay area), and the processor stack are used by debug operations. (See Target Application Notes.)

Establish Debug Control and Set a Breakpoint

Load DEMO3.TSK, and use the main menu's **F6** to get to the debug sub-menu.

The display shows the functions that are available in menu mode. [figure 5-2a]

Press **F1** to establish debug control.

Debug control must be established before other debug options can be used. Here, the program requests a breakpoint address.

Type **88** and a carriage return to set the address.

The equivalent command line RESET 88 RB establishes debug control at address 0088. (RESET enables the target system to be reset by RB.) [figure 5-2b]

The breakpoint display shows the registers and their contents, prior to the effects of the current instruction, followed by the address, opcode, and disassembly.

Here, the contents of DE and HL have changed since initialization because of the block-move operation. IX and IY hold arbitrary values not used by DEMO3.TSK. The "A" byte of AF contains the 01 to be output by the current operation. (F is the flag register: its bits are shown to the right in parentheses. A capital letter means the bit is set, lower-case means it is reset, and a hyphen means the bit is unused. See the Target Application Notes.)

Single-Step the Target System

Press **F3** to execute the next step of the program.

The breakpoint display shows unchanged register contents, and the RRCA instruction.

Press **F3** again.

Note that the AF value has changed (because of the RRCA), and that the current instruction calls the delay loop at A0.

Press **F3** once more.

The N command issued by F3 "falls through" loops and branches. Now the breakpoint display shows the processor's state after its return from the delay loop. The current instruction jumps to the main program loop.

Follow Jumps and Loops

To follow the jump, press **F4**.

As the address shows, the NMI command followed the program's jump instruction.
[figure 5-4a]

Type **LP** to execute the entire loop once.

The lighted LED advances one position with each iteration of this loop. LP is a quick way to execute a loop once and stop at the same address. LP only works when the program is stopped at an address that will be executed again.
[figure 5-4b]

Alter Register Contents

DEMO3.TSK rotates a single bit in register A, using it to animate the microtarget's LEDs. While at a breakpoint, you can change the register's contents to a new bit pattern and see the effect. Here we will put a different value into A just before the output/rotate sequence.

If you followed the instructions above, your breakpoint display should show address 0088, with the OUT (79),A instruction that will output the current contents of A. We will not want to change the value of the flags register (F) at this time, so type **7Fxx =AF** <Enter>, where xx is the current value of F in your breakpoint display. This replaces the current value of A with 7F.

Enter the command **RZ** to resume execution.

The microtarget resumes execution with the new register contents intact, and without resetting or restarting the program. The LEDs now exhibit a new pattern, even though you have not modified the program itself.

Now, with the program executing, press **F4**.

This sends a non-maskable interrupt to the target processor. It gains debug control of the running system within a few cycles, and generates a breakpoint display. When you are already at a breakpoint, NMI single-steps through normal program flow.

Resume Execution to a New Breakpoint

Press **F2** to resume execution from the current breakpoint until a new breakpoint is reached.

Respond with **A0** as the new breakpoint address.

The equivalent command is A0 RB. The breakpoint display is shown when that address is reached.

RB, unless it is preceded by RESET, allows a program to continue execution with a new breakpoint set.

For other ways to resume execution after establishing debug control, use function key choices from the analyzer and debug menus, or directly enter:

RZ	to resume execution from the current state.
<adr> **G**	to exit debug control and resume execution from any address.
<adr-a> <adr-b> **GB**	to resume execution at address-a, with a breakpoint set for address-b.
<adr> **GW**	to go to an address and wait for an analyzer command.
STARTUP	to reset and start the microtarget and the analyzer.
NORMx <trig> **S**	to restart the analyzer and reset the target (if RESET is

enabled), providing a trace of execution when a trigger appears.

Define Debug Triggers

Earlier, you learned a set of commands to start the analyzer at any point during target execution. The same trigger-specification commands can set a breakpoint. When so used, triggers issue an NMI to the target processor, instead of affecting the trace buffer. (For debugging purposes, triggers set a qualifier internally, so AFTER should not be used.)

Type **RI**, the debug equivalent of the analyzer's NORMx words.

Enter **79 LADR 20 DATA SI** to interrupt the processor after 20 is sent to port 79.

The RI ... SI sequence gains debug control within a cycle or two after the specified trigger event. Here, A has already been rotated and the current instruction calls the delay loop. SI is the debug version of S.

[figure 5-6]

Display the Alternate Registers

The Z80's alternate register set can be included in the breakpoint display.

Type **SHOW-ALT**.

Now enter **RESET 88 RB** (or use F1 in the debug menu).

The display now shows the contents of the alternate registers.

Type **SHOW-ALT'** and press **F3** to execute the next instruction.

The display returns to its default state, without the alternate register display.

Read and Write to Target Ports

The sample program, DEMO3.TSK, controls the microtarget's LEDs by sending values to port 79. Two commands make it possible to read a port's contents, and to send a new value to a port.

With DEMO3.TSK loaded, use **STARTUP** to get the program going.

Type **NMI** to gain debug control.

Enter **79 INP** <carriage return> to see the hex contents of port 79.
    The resulting number is the value that set the current LED pattern.

To try different port output, type **AA 79 OUT**.
    The LEDs now light up according to the bit pattern of AA hex.
    Experiment with other values to see their effects.
    [figure 6-6]

Experiment by setting some breakpoints and single-stepping through the target system's execution. You can gain debug control at any address that contains the first byte of an opcode. If use an invalid address as a breakpoint, the system will wait indefinitely – when this happens, press the Enter key to generate a hardware interrupt.

OPERATE ON MEMORY

Disassemble From Memory
Set a starting memory address
Choose length of disassembly

Examine and Alter Hex Memory Dump
Select a starting memory address
Move through the display
Edit hex or ASCII values
Save the changes

Operate on Memory

The UniLab includes several tools that work directly on memory. You can disassemble the contents of memory to get an assembly code listing, or use a hex/ASCII editor to examine and change memory.

Disassemble a Program

We will begin looking at memory operations with a simple disassembly of the program in emulation ROM. Load DEMO3.TSK, as described earlier.

At the main menu, press **F3** to display the sub-menu of memory operations.

Use **F2** to disassemble from memory.
    As the program prompts you for input, enter **0** as the starting address and **7** for the number of (hex) lines you wish to disassemble.
[figure 6-2]
    The program displays the equivalent keyboard command 0 7 DM, followed by the disassembly. These are the initialization instructions, the block move, and a jump instruction to the program's main routine, followed by unused addresses.

Now press **F2** again, entering **82** for the starting address and **7** for the number of lines.
    Here is the DEMO3.TSK code that controls the LEDs.
    - After initializing the port chip's control register (I/O address 7B),
    - it loads a value into A at address 0086,
    - sends that value to port "A" of the port chip (I/O address 79),
    - rotates A's contents, calls the delay loop at 00A0, and
    - jumps back to address 0088 to output the new bit pattern from A.

Any address can be the starting point of a disassembly, and there is no limit to the number of lines. You can disassemble emulation memory and, with debug control, target RAM. Using DM from the command mode (or a special window, to be discussed later) allows more lines to fit on the screen.

Examine and Alter Hex Memory Dump

UniLab's hex/ASCII editor permits you to examine and alter blocks of memory, and to rapidly page through consecutive blocks.

From the main menu, press **F3** for the sub-menu of memory operations.

Press **F1** and, when prompted for it, the starting address **0**.

The command 0 MODIFY is executed, bringing a display of the first 90 bytes of memory. The hex display is shown alongside the ASCII display. The bottom line shows which keys control the editor.

The first 18 bytes show the opcodes and values of the initialization routine you disassembled previously, ending with the jump to 0082.

Use the **left-arrow** and **right-arrow** keys to move the cursor.

The cursor moves from address to address, automatically wrapping around line endings.

Use the **down-arrow** key to move down the rows.

If you scroll past the last line, the contents of additional addresses will be shown.

Now use the **up-arrow** to move above the line containing address 0.

A message states that you have entered target RAM, then the new address range is displayed. If you access target RAM without first establishing debug control, UniLab issues a non-maskable interrupt to get control of the necessary resources.

PgDn and **PgUp** display consecutive blocks of memory.

Program instructions can be changed with the memory editor, but you will use it more often to modify data, to try a new value when initializing a register or variable, or to edit text strings.

Use **Ctrl-Right Arrow** to move the cursor into the ASCII display.
Enter any ASCII characters here, and their hex equivalents are shown to the left. Ctrl-Left Arrow moves the cursor back to the hex display.

Press **Esc** to leave the editor without saving this input.
[figure 6-3]
Re-enter the editor by pressing **F1** and responding with address **0**.

Move the cursor to A2, where the delay loop's length is stored.

Type **10** to replace the 40.
This alters the value that is decremented in this routine, effectively making the delay loop shorter.

Press **End** to save the change in program memory.
UniLab responds with "Changes saved." The LEDs blink much faster than before. If you wish, try other values at 00A2.

Changes to memory remain in effect until you change them again, or reload and restart the program. When you want a printed record of your changes to memory, the mode panel discussed later provides an option that will echo the CRT display to the printer.

LINE-BY-LINE ASSEMBLER

Alter the Target Code

Invoke the assembler at an address

Use assembly language

Overwrite program memory

Write a Patch Routine

Assign a patch address

Write the new code

Jump to the patch

Line-by-Line Assembler

You can edit code and write small test routines or patches quickly with UniLab's assembler, bypassing the compile-assemble-link process. You can also use it to be certain a value appears on the bus. The assembler supports all Z80 instructions (except the * location counter), arithmetic expressions, and symbols.

Alter the Target Code

The sample program begins by moving a block of memory from 0100 to 1800, but that code is never executed unless you alter the instruction at address 008E so that it jumps to 1800 instead of 0088. This would be easy to change with the memory editor, but here we will use the line-by-line assembler.

The assembler is only available in command mode. Start by loading the DEMO3.TSK program (0 FFF BINLOAD DEMO3.TSK).

Type **8E ASM** to invoke the assembler, starting at address 008E.

The program displays the requested address 008E, with the cursor indicating where the new assembly language instruction may be entered.

Type **JP 1800** <Enter>.

This will assemble into jump to address 1800. The assembler now offers the next address for any further changes.
[figure 7-3a]

Press **Enter** to leave the contents of address 0091 unchanged and exit the assembler. Use **STARTUP** to execute the altered program.

The LED sequence changes in accord with the new output routine.

The analyzer can provide a filtered trace showing the new program flow. To do so, turn off reset with RESET' (to skip the initialization sequence) and type ONLY NOT A0 TO A9 ADR S. You will see that the routine at 1800:
- outputs 1 from A to the target port,
- rotates the bit pattern in A (making the 1 an 80),
- calls the delay loop at A0,
- then jumps to 0088 to output the 80,
- and ends up back at 1800 again.

Write a Patch Routine

The assembler overwrites whatever is in memory. Extra instructions cannot be inserted between existing addresses, but you can put in a jump to any unused area of memory and write a multi-line patch there.

For example, you can write code to brings a register's contents onto the bus. Then that value, which would not appear on the bus otherwise, can be used to trigger a trace.

Type **88 ASM** to write new code beginning at address 0088.
At the 0088, type **CALL 300** <Enter>
At the 008B, press <Enter>

Now type **300 ASM** <Enter>
At the 0300, type **OUT (79)A** <Enter>

> 0302 **RRCA** <Enter>
> 0303 **PUSH AF** <Enter>
> 0304 **POP AF** <Enter>
> 0305 **RET** <Enter>
> 0306 <Enter>
> [figure 7-3b]
> The UniLab assembler provides the address numbers. A carriage return on an empty line preserves the previous contents of that address.

The above patch puts the contents of AF onto the bus before the delay loop is called and before A is sent to the target port. To trigger a breakpoint when the value 20 is in A:

Type **RI 20 DATA AFTER 303 ADR ADR SI.**

> The breakpoint display shows 2060 in AF, and the microtarget's second LED is lit.

Type **LP** to execute the loop once, and you will see that the 20 in A causes the third LED to light, and that the RRCA instruction changed A's value to 10.

Use Symbolic Labels

Define symbols within UniLab
Load a symbol file
List current symbols
Use symbolic triggers and breakpoints

Use Symbolic Labels

UniLab allows you to load symbol tables generated by your linker. It supports a wide variety of symbol table formats. After loading a symbol table, the analyzer will display the same labels as the source code. Or, if you wish, you can define symbols from within the UniLab program. Triggers and breakpoints are easier to set, and a trace display is more quickly understood, with symbolic names.

Load the DEMO3.TSK program and use STARTUP to generate a trace display.

Define Symbols Within UniLab

Type **3 IS INIT_IX** and **1234 IS VALUE_IX**.
[figure 7-4]
Regenerate the trace display by pressing **Home**.

> A new column is added to the display, where the symbolic name INIT_IX is shown beside address 0003. While the opcode for that address contains the value 1234, the mnemonics column shows VALUE_IX.

Load a Symbol File

You will usually load a target program's original symbol table, rather than assigning many new symbols individually:

Type **SYMFILE** and, at the prompt for a filename, type **DEMO3.SYM**.

> This calls up the Symbol File Format menu – choose **F5**. To see a list of the symbols that are now resident, type **SYMLIST**.

Press **Home** to see the top of the trace display again.

> [figure 7-5]

Type **NORMT OUT_LOOP ADR S** to trigger the analyzer on the output loop.

When symbolic labels have been defined, you can use them with UniLab commands anywhere you would type in a value to examine memory (as in DELAY_LOOP MODIFY), or to set triggers and breakpoints. For added convenience during development work, refer to the related commands SYMSAVE, SYMFILE+, and SYMLOAD in the on-line glossary (type HELP <command>).

CONTROL THE UNILAB ENVIRONMENT

Control the displays
Increase productivity

Open Windows on Your Work
Use general-purpose windows
Generate a disassembly window
Change window sizes

Use Function Keys
Display function key assignments
Reassign function keys

Define Macro Commands
Enable macro functions
Define a macro
Assign macros to function keys
Disable macro functions

Use the Mode Panel
Adjust the analyzer, display, and log modes

Save Your Changes to the System

Control the UniLab Environment

The features presented here affect the way UniLab displays information and responds to input. You can modify the system's default settings to suit your particular needs and work style.

Open Windows on Your Work

The display screen can be divided into "windows." You can refer to a disassembly and a trace display at the same time, or keep a portion of one trace on the screen and generate a new one. You must leave menu mode in order to use screen windows, but call up the menus again any time you wish.

Load the DEMO3.TSK program and enter **RESET 88 AS** to trigger the analyzer at address 0088.

> The screen fills with a trace display.

Press **F2** once to split the screen horizontally.

> A line shows the screen division, and the lower half is cleared. The upper window retains its contents. Press **End** to move the cursor between windows.
>
> In command mode, F2 toggles the split-screen display on and off. The same analyzer settings, triggers, and debug options are in effect, regardless of which window you use.

Press **PgDn** to scroll through the trace buffer.

> One half of the screen remains unchanged, while the window with the cursor responds to your keystrokes.

In either window, type **RESET 88 RB**.

> The active window shows the breakpoint display, while the other window remains unchanged.

Press **F2** restore the full screen.

These "general purpose" windows are easily used with most commands. A special-purpose disassembly window will remain on the screen while you perform other functions.

Type **0 DN** to disassemble memory, starting from address 0000.

A vertical rule delineates the right one-third of the screen, which displays the requested listing. Enough memory is disassembled to fill the window.

Press **F2** to split the screen, and type **A0 DN** to see more disassembled memory.

The new disassembly is restricted to the height of the current window. [figure 8-2]

The contents of the disassembly window remain in place, until you use:

- a new **DN** command,
- **F2** to toggle the split-screen display,
- **End** to move the cursor between windows, or
- **F10** to call up the main menu.

You can use up to four windows to visually compare:

Target execution
Disassembled code
Register displays
Memory contents
Help and other text files

Help files will not be shown in their full width if there is an active disassembly window. Use F2 or End to adjust the display appropriately.

Change the Window Sizes

Window dimensions can be modified. Perhaps you would like a narrower disassembly window, or a window that is only three or four lines high for breakpoint displays.

Press **Shift-F8** or type **WSIZE**.

The screen temporarily clears, and the current window borders are displayed. Now use the **arrow keys** to adjust the dimensions of the windows.

When you are satisfied, press **End** to accept the changes.

The display is returned to its former state. Any new commands entered will reflect the new window sizes.

Use Function Keys

UniLab uses function keys for easy access to often-used commands, which can save a great many keystrokes. Function keys can be used alone, or with the Shift, Alt, or Ctrl keys. Not all have been pre-assigned, and all the keys are reassignable.

To see what functions are assigned to the keys, use **F1**, **SHIFT-F1**, **ALT-F1**, and **CTRL-F1**. It is easy to change or add function-key assignments, and any such changes will be shown in the displays.
[figure 8-5]

Reassign Function Keys

Type **6 FKEY PINOUT** to assign the PINOUT command to F6.
Now, pressing F6 will display the pinout of your processor.
[figure 8-6]

You can assign a command to any function key by using the words FKEY, SHIFT-FKEY, ALT-FKEY, or CTRL-FKEY. Their syntax is the same:

```
<key#>  FKEY  <command>
```

It is possible to assign strings of commands to a single function key, saving even more keystrokes. To do so, you must first learn to define a "macro" function that executes other commands. Then that macro's name can be assigned to a function key. (See entries in the on-line glossary for : and MACRO.)

Define Macro Commands

If you type the same input often, consider defining it as a macro. When you enter a complex trigger spec repeatedly, for example, making it a one-word macro will save keystrokes and minimize typing errors. Macros can use UniLab commands, data, and even commands from the underlying programming language. Each macro should have a unique name of any length (no spaces are allowed).

To invoke these capabilities, type **MACRO** and a carriage return.

After a few seconds of disk access to make macro functions and the programming language available, the system tells you that changes to the system can be made permanent with the SAVE-SYS command (see below).

Now, suppose you use a particular trigger specification often. In DEMO3TSK, for example, it was quite useful to filter the delay loop from the trace:

Type : **TRIGGER1 NORMT ONLY NOT A0 TO A9 ADR** ; <Enter>

Now type **TRIGGER1** to enter the trigger spec, and **S** to start the analyzer.

The trace is displayed in accordance with the trigger specifications, but without requiring all the parameters to be retyped each time the analyzer settings have been changed.

[figure 8-8]

Use the Mode Panel

A three-part "mode panel" provides control of UniLab's default settings. You can simplify the analyzer display, log a session to disk or printer, and disable the hardware interrupt, among other options.

Press **F8** to see the first page of the mode panel.

> The panel appears in the upper-right corner of the screen. This is the first of three pages that correspond to analyzer, display, and log modes. The down- and up-arrows move the cursor from item to item.

[figure 8-9]
If you do not need the disassembly, control, or miscellaneous inputs, that data can be removed easily from the display:

On the mode panel's first page, with the cursor beside "DISASSEMBLER," press the **right-arrow** to turn off the disassembler.

> The disassembler always tries to interpret trace data as if it were from consecutive cycles. In a narrowly filtered trace, that can be inappropriate. At such times, turn off the disassembler to avoid confusing, incorrect mnemonics.

Press **PgDn** to move to the panel's next page, then use the **arrow keys** to turn off the MISC and CONT columns.

Press **End** when you are done, and use STARTUP to start the analyzer; the trace reflects the changes you made by suppressing the CONT, MISC, and mnemonics columns.

You can press **F8** to return to the mode panel and restore the default settings.

Save Your Changes to the System

Whenever you alter the default UniLab environment and wish to make the change permanent, type SAVE-SYS before exiting the program and provide the filename you will use to load the program (e.g., ULZ80MOD). The new settings will be in effect the next time you use UniLab. Users of floppy-based systems will have to save the new filename to drive B or save their settings in the original ULZ80 file.